浙江大学

计算机视觉(本科)作业报告

作业名称:	**Eigenface**
姓 名:	曹小川
学 号:	3200105705
电子邮箱:	434446127@qq.com
联系电话:	18991060018
导 师:	潘纲

2022年 12 月 8 日

Eigenface

一、 作业已实现的功能简述及运行简要说明

作业已实现功能：写代码实现Eigenface人脸识别的训练、识别、重构过程，利用公开的人脸数据集构建一个人脸库，包括自己的人脸模型。

二、 作业的开发与运行环境

开发工具:Visual Studio

操作系统: Windows 10

编程语言：C++/ OpenCV

三、 系统或算法的基本思路、原理、及流程或步骤等

首先实现对人脸模型的训练，读入人脸库每人前五张图片，对其进行加载储存，然后计算协方差矩阵，均值矩阵，并利用协方差矩阵计算特征值以及特征向量，根据输入的`energyPercent`确定需要利用特征矩阵的尺寸，根据此来形成特征脸，训练模型。在测试过程中，首先输入训练好的模型，然后根据用户的输入来确定对哪一个人的图片进行检测，通过计算输入图像在模型上的投影与训练集中图像的距离来判断输入的图片是哪个人的。在重构阶段，对输入的人脸图像进行变换到特征脸空间，然后再用变换后的结果重构回原来人脸图像。

四、 具体如何实现，例如关键（伪）代码、主要用到函数与算法等

主要数据结构

```
class face {
public:
    string img_Path;
    string eye_Path;
    Mat src;
    Mat gray;
    Mat handle;
    Mat trans;
    Mat eq;
    Mat vect;
    int eyelx, eyely, eyerx, eyery;
    string person_id;
    void load_Img(string _imgpath, string _eyepath);
    void load_Eye(string path);
};
```

用来加载一张图片，储存其信息。

```
class facelib {
public:
    int Size;
    vector<face*>entrys;
    vector<Mat> _samples;
    vector<Mat> sam;
```

```
    Mat samples;

    void load(string pathi ,string pathe);
    ~facelib() {
        for (int i = 0; i < entrys.size(); i++) {
            face* f = entrys.back();
            entrys.pop_back();
            delete(f);
        }
    }
};
```

加载所有的图片，储存所有`face`结构的信息。

主要函数

```
void mytrain() {
    string model_name = "model";
    double energy = 0.95;
    cout << "Read in facelib..." << endl;
    fl.load(picpath,eyepath);
    Mat samples, mmat ,cmat; //输出的协方差矩阵,均值矩阵
    fl.samples.copyTo(samples);
    cout << samples.size() << endl;
    cout << "Caculating Covariance Mat..." << endl;
    calcCovarMatrix(fl.sam, cmat, mmat, CV_COVAR_ROWS | CV_COVAR_NORMAL);
    Mat meanImg;
    mmat.convertTo(meanImg, CV_8UC1);
    imwrite("mean_img.png", meanImg);
    cout << "cmat" << cmat.type() << endl;
    cout << "cmat" << cmat.size() << endl;
    cout << "mmat" << mmat.size() << endl;
    Mat eigenVectors = Mat();
    Mat eigenValues = Mat();
    vector<Mat> eigenFaces;
    eigen(cmat, eigenValues, eigenVectors);
    cout << eigenVectors.size() << endl;
    double energyPercent = 0.95;
    int eigenCount = (int)(eigenVectors.rows * energyPercent);
    cout << "eigen count: " << eigenCount << endl;
    Mat AT = Mat(eigenCount, standardSize.height * standardSize.width, CV_64F);
    Mat A = Mat(standardSize.height * standardSize.width, eigenCount, CV_64F);//
特征向量
    for (int i = 0; i < eigenCount; i++) {
        Mat t = Mat(standardSize.height, standardSize.width, CV_64F);
        Mat tt = Mat(standardSize.height, standardSize.width, CV_8UC1);
        for (int j = 0; j < standardSize.width * standardSize.height; j++) {
            t.at<double>(j / standardSize.width, j % standardSize.width) =
eigenVectors.at<double>(i, j);
            AT.at<double>(i, j) = eigenVectors.at<double>(i, j);
            A.at<double>(j, i) = eigenVectors.at<double>(i, j);
        }
        normalize(t, t, 255, 0, NORM_MINMAX);
        t.convertTo(tt, CV_8UC1);
        eigenFaces.push_back(tt);
```

```
    }

    cout << "Write Model..." << endl;
    FileStorage model(model_name, FileStorage::WRITE);
    model << "eigenCount" << eigenCount;
    model << "allFaces" << fl.sam;
    model << "A" << A;
    model << "eigenVectors" << eigenVectors;
    model << "mean" << mmat;
    model.release();

    Mat tmp1,tmp,tmp2;
    vector<Mat> t1, t2,t;
    for (int i = 0; i < 5 && i < eigenCount; i++) {
        t1.push_back(eigenFaces.at(i));
    }
    hconcat(t1, tmp1);
    t.push_back(tmp1);
    cout << 123 << endl;
    for (int i = 5; i < 10 && i < eigenCount; i++) {
        t2.push_back(eigenFaces.at(i));
    }
    hconcat(t2, tmp2);
    t.push_back(tmp2);
    vconcat(t, tmp);
    imwrite("top10_img.png", tmp);
    imshow("Top10", tmp);
    waitKey(0);
}
```

训练模型的主要函数，计算协方差矩阵，特征值，特征向量，计算储存模型，生成Top的特征脸等。

```
void face::load_Img(string _imgpath, string _eyepath) {
    img_Path = _imgpath;
    eye_Path = _eyepath;
    load_Eye(_eyepath);
    src = imread(img_Path);
    cvtColor(src, gray, COLOR_RGB2GRAY);
    Point mid;
    mid.x = (eyelx + eyerx) / 2;
    mid.y = (eyely + eyery) / 2;
    double angle = atan((eyery - eyely) * 1.0 / (eyerx - eyelx) * 1.0) * 180 /
CV_PI;
    handle = getRotationMatrix2D(mid, angle, 1.0);
    warpAffine(gray, trans, handle, gray.size());
    equalizeHist(trans, trans);//直方图均衡化是通过拉伸像素强度分布范围来增强图像对比度的
一种方法
    resize(trans, trans, standardSize);
    trans.copyTo(eq);
    vect = eq.reshape(1, 1).t();//转置
    string name="";
    person_id = getIDfrompath(_imgpath);
}
```

加载图片

```
void face::load_Eye(string path) {
    ifstream file(path.c_str());
    string s1, s2, s3, s4;
    string Read;
    if(!file){
        cout << "Can Load Eye file" << endl;
        return;
    }
    string line1;
    string line2;
    getline(file, line1);
    getline(file, line1);

    getline(file, line1);
    for (int i = 0; i < line1.size(); i++) {
        if (line1[i] >= '0' && line1[i] <= '9') {
            eyelx = eyelx * 10 + line1[i] - '0';
        }
    }
    getline(file, line2);
    for (int i = 0; i < line2.size(); i++) {
        if (line2[i] >= '0' && line2[i] <= '9') {
            eyely = eyely * 10 + line2[i] - '0';
        }
    }
    getline(file, line1);
    getline(file, line1);
    getline(file, line1);
    for (int i = 0; i < line1.size(); i++) {
        if (line1[i] >= '0' && line1[i] <= '9') {
            eyerx = eyerx * 10 + line1[i] - '0';
        }
    }
    getline(file, line2);
    for (int i = 0; i < line2.size(); i++) {
        if (line2[i] >= '0' && line2[i] <= '9') {
            eyery = eyery * 10 + line2[i] - '0';
        }
    }
    file.close();
}
```

从json文件中获取眼部信息。

```
void facelib::load(string pathi,string pathe) {
    vector<String> img1, img2, img3, img4, img5,img;
    vector<String> eye1, eye2, eye3, eye4, eye5,eye;
    glob(pathi + "/1.pgm", img1, true);
    glob(pathe + "/1.json", eye1, true);
```

```
    glob(pathi + "/2.pgm", img2, true);
    glob(pathe + "/2.json", eye2, true);
    glob(pathi + "/3.pgm", img3, true);
    glob(pathe + "/3.json", eye3, true);
    glob(pathi + "/4.pgm", img4, true);
    glob(pathe + "/4.json", eye4, true);
    glob(pathi + "/5.pgm", img5, true);
    glob(pathe + "/5.json", eye5, true);

    img.insert(img.end(), img1.begin(), img1.end());
    img.insert(img.end(), img2.begin(), img2.end());
    img.insert(img.end(), img3.begin(), img3.end());
    img.insert(img.end(), img4.begin(), img4.end());
    img.insert(img.end(), img5.begin(), img5.end());

    eye.insert(eye.end(), eye1.begin(), eye1.end());
    eye.insert(eye.end(), eye2.begin(), eye2.end());
    eye.insert(eye.end(), eye3.begin(), eye3.end());
    eye.insert(eye.end(), eye4.begin(), eye4.end());
    eye.insert(eye.end(), eye5.begin(), eye5.end());

    if (img.size() == 0 || eye.size() == 0) {
        cout << "no such files" << endl;
        return;
    }
    else if (img.size() != eye.size()) {
        cout << "imgs and eyes are not fit" << endl;
        return;
    }

    sort(img.begin(), img.end());
    sort(eye.begin(), eye.end());

    Size = img.size();
    for (int i = 0; i < img.size(); i++) {
        face *f = new face();
        f ->load_Img(img.at(i), eye.at(i));
        entrys.push_back(f);
        sam.push_back(f->trans);
        _samples.push_back(f->vect);//列向量
    }
    hconcat(_samples, samples);
    cout << "Size of facelib is " << Size << endl;
    Mat m;
    samples.convertTo(m, CV_8U);
    imwrite("face0_trans.png", entrys.at(0)->trans);

}
```

加载所有待训练的图片。

```
void mytest() {
    string model_name = "model";
```

```
    cout << "Read in Model..." << endl;
    FileStorage model(model_name, FileStorage::READ);
    Mat A;
    vector<Mat>allFaces;
    vector<Mat>src;
    vector<Mat>dst;
    int todet;
    int eigenCount;
    model["A"] >> A;
    model["allFaces"] >> allFaces;
    model ["eigenCount"] >> eigenCount;
    cout << "input the test object:" << endl;
    cin >> todet;
    for (int i = 6; i <= 10; i++) {
        string ima_path = "att-face/s" + to_string(todet) + "/" + to_string(i) +
".pgm";
        string eye_path = "att-eye-location/s" + to_string(todet) + "/" +
to_string(i) + ".json";
        face* f = new face();
        f->load_Img(ima_path, eye_path);
        Mat testImg = f->trans;
        src.push_back(testImg);
        Mat testDoubleMat;
        testImg.reshape(0, 1).convertTo(testDoubleMat, CV_64F);
        Mat testCoordinates = testDoubleMat * A;
        double minDistance = -1;
        face resultFace;
        int id = 0;

        for (auto iter = allFaces.begin(); iter != allFaces.end(); iter++, id++)
{
            double distance;
            face f_test;
            f_test.trans = *iter;
            Mat doubleMat;
            f_test.trans.reshape(0, 1).convertTo(doubleMat, CV_64F);
            f_test.eq = doubleMat * A;
            distance = 0;
            for (int i = 0; i < eigenCount; i++) {
                distance += pow(f_test.eq.at<double>(0, i) -
testCoordinates.at<double>(0, i), 2);
            }
            if (distance < minDistance || minDistance == -1) {
                minDistance = distance;
                resultFace = f_test;
            }
        }
        dst.push_back(resultFace.trans);
    }
    cout << "haha" << endl;
    Mat m1, m2, m;
    vector<Mat> fi;
    vconcat(src, m1);
    vconcat(dst, m2);
    cout << "haha" << endl;
```

```
    fi.push_back(m1);
    fi.push_back(m2);
    hconcat(fi, m);
    imwrite("test result.png", m);
    imshow("test", m);
    waitKey(0);
}
```

检测输入图片属于训练集哪一个人。

根据用户的输入来确定对哪一个人的图片进行检测，通过计算输入图像在模型上的投影与训练集中图像的距离来判断输入的图片是哪个人的。

```
void myreconstruct() {
    string model_name = "model";
    cout << "Read in Model..." << endl;
    FileStorage model(model_name, FileStorage::READ);
    Mat A;
    Mat eigenVectors = Mat();
    Mat mean;
    vector<Mat> allFaces;
    vector<Mat> tmp;
    Mat res;
    int eigenCount;
    model["A"] >> A;
    model["allFaces"] >> allFaces;
    model["eigenCount"] >> eigenCount;
    model["eigenVectors"] >> eigenVectors;
    model["mean"] >> mean;
    face* f = new face();
    int todet;
    int ph;
    cout << "input the person_id" << endl;
    cin >> todet;
    cout << "input the photo_id" << endl;
    cin >> ph;
    string ima_path = "att-face/s" + to_string(todet) + "/" + to_string(ph) +
".pgm";
    string eye_path = "att-eye-location/s" + to_string(todet) + "/" +
to_string(ph) + ".json";
    f->load_Img("att-face/s0/1.pgm", "att-eye-location/s1/1.json");
    Mat testDoubleMat;
    Mat testImg = f->trans;

    testImg.reshape(0, 1).convertTo(testDoubleMat, CV_64F);
    for (int i = 0; i < 5 ;i++) {
        int num_components;
        if (i == 0) num_components = 10;
        else if (i == 1) num_components = 25;
        else if (i == 2) num_components = 50;
        else if (i == 3) num_components = 75;
        else num_components = 585;
        Mat evs = Mat(A, Range::all(), Range(0, num_components));
```

```
        Mat testCoordinates = testDoubleMat * evs;
        Mat recst = testCoordinates * (evs.t());
        normalize(recst.reshape(1, f->trans.rows), recst, 255, 0, NORM_MINMAX);
        recst.convertTo(recst, CV_8UC1);
        tmp.push_back(recst);
    }
    hconcat(tmp, res);
    imwrite("reconstruct.png", res);
    imshow("res", res);
    waitKey(0);

}
```

对输入图片进行重构。

对输入的人脸图像进行变换到特征脸空间，然后再用变换后的结果重构回原来人脸图像。

五、 实验结果与分析

程序代码：

```
//main.cpp

#include"facelib.h"
using namespace cv;
using namespace std;
facelib fl;

int main() {
    int type = 0;
    cout << "input the type: \ntrain:1\ttest:2\treconstruct:3" << endl;
    cin >> type;
    if (type == 1) {
        mytrain();
    }
    else if (type == 2) {
        mytest();
    }
    else if (type == 3) {
        myreconstruct();
    }
    else {
        cout << "invalid type" << endl;
    }
}
```

```
//facelib.h
#pragma once
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui_c.h>
#include <fstream>
#include <iostream>
#include <string>
```

```
#include <vector>
#include <stdlib.h>
#include <sstream>
using namespace std;
using namespace cv;
const int WIDTH = 25;
const int HEIGHT = 25;
class face {
public:
    string img_Path;
    string eye_Path;
    Mat src;
    Mat gray;
    Mat handle;
    Mat trans;
    Mat eq;
    Mat vect;
    int eyelx, eyely, eyerx, eyery;
    string person_id;
    void load_Img(string _imgpath, string _eyepath);
    void load_Eye(string path);
};

class facelib {
public:
    int Size;
    vector<face*>entrys;
    vector<Mat> _samples;
    vector<Mat> sam;
    Mat samples;

    void load(string pathi ,string pathe);
    ~facelib() {
        for (int i = 0; i < entrys.size(); i++) {
            face* f = entrys.back();
            entrys.pop_back();
            delete(f);
        }
    }
};

void mytrain();
void mytest();
void myreconstruct();
Mat toImg(Mat vec, int w, int h);
string getIDfrompath(string s);
```

```
//mytrain.cpp
#include"facelib.h"


string picpath = "att-face";
string eyepath = "att-eye-location";
Size standardSize = Size(25, 25);
extern facelib fl;
```

```
string getIDfrompath(string s) {
    string id = "";
    for (int i = 0; i < s.length(); i++) {
        while (s[i] != 's' && i < s.length()) { i++; }
        while (s[i] != '\\' && i < s.length()) {
            id += s[i];
            i++;
        }
        break;
    }
    return id;
}

Mat toImg(Mat vec, int w, int h){
    assert(vec.type() == 6);
    assert(vec.cols == w * h);
    Mat res(Size(w, h), CV_64FC1);
    for (int i = 0; i < h; i++) {
        vec.colRange(i * w, (i + 1) * w).convertTo(res.row(i), CV_64FC1);//转化为
一列
    }
    normalize(res, res, 1.0, 0.0, NORM_MINMAX);
    return res;
}
void mytrain() {
    string model_name = "model";
    double energy = 0.95;
    cout << "Read in facelib..." << endl;
    fl.load(picpath,eyepath);
    Mat samples, mmat ,cmat; //输出的协方差矩阵,均值矩阵
    fl.samples.copyTo(samples);
    cout << samples.size() << endl;
    cout << "Caculating Covariance Mat..." << endl;
    calcCovarMatrix(fl.sam, cmat, mmat, CV_COVAR_ROWS | CV_COVAR_NORMAL);
    Mat meanImg;
    mmat.convertTo(meanImg, CV_8UC1);
    imwrite("mean_img.png", meanImg);
    cout << "cmat" << cmat.type() << endl;
    cout << "cmat" << cmat.size() << endl;
    cout << "mmat" << mmat.size() << endl;
    Mat eigenVectors = Mat();
    Mat eigenValues = Mat();
    vector<Mat> eigenFaces;
    eigen(cmat, eigenValues, eigenVectors);
    cout << eigenVectors.size() << endl;
    double energyPercent = 0.95;
    int eigenCount = (int)(eigenVectors.rows * energyPercent);
    cout << "eigen count: " << eigenCount << endl;
    Mat AT = Mat(eigenCount, standardSize.height * standardSize.width, CV_64F);
    Mat A = Mat(standardSize.height * standardSize.width, eigenCount, CV_64F);//
特征向量
    for (int i = 0; i < eigenCount; i++) {
        Mat t = Mat(standardSize.height, standardSize.width, CV_64F);
        Mat tt = Mat(standardSize.height, standardSize.width, CV_8UC1);
```

```
        for (int j = 0; j < standardSize.width * standardSize.height; j++) {
            t.at<double>(j / standardSize.width, j % standardSize.width) =
eigenVectors.at<double>(i, j);
            AT.at<double>(i, j) = eigenVectors.at<double>(i, j);
            A.at<double>(j, i) = eigenVectors.at<double>(i, j);
        }
        normalize(t, t, 255, 0, NORM_MINMAX);
        t.convertTo(tt, CV_8UC1);
        eigenFaces.push_back(tt);
    }

    cout << "Write Model..." << endl;
    FileStorage model(model_name, FileStorage::WRITE);
    model << "eigenCount" << eigenCount;
    model << "allFaces" << fl.sam;
    model << "A" << A;
    model << "eigenVectors" << eigenVectors;
    model << "mean" << mmat;
    model.release();

    Mat tmp1,tmp,tmp2;
    vector<Mat> t1, t2,t;
    for (int i = 0; i < 5 && i < eigenCount; i++) {
        t1.push_back(eigenFaces.at(i));
    }
    hconcat(t1, tmp1);
    t.push_back(tmp1);
    cout << 123 << endl;
    for (int i = 5; i < 10 && i < eigenCount; i++) {
        t2.push_back(eigenFaces.at(i));
    }
    hconcat(t2, tmp2);
    t.push_back(tmp2);
    vconcat(t, tmp);
    imwrite("top10_img.png", tmp);
    imshow("Top10", tmp);
    waitKey(0);
}

void face::load_Img(string _imgpath, string _eyepath) {
    img_Path = _imgpath;
    eye_Path = _eyepath;
    load_Eye(_eyepath);
    src = imread(img_Path);
    cvtColor(src, gray, COLOR_RGB2GRAY);
    Point mid;
    mid.x = (eyelx + eyerx) / 2;
    mid.y = (eyely + eyery) / 2;
    double angle = atan((eyery - eyely) * 1.0 / (eyerx - eyelx) * 1.0) * 180 /
CV_PI;
    handle = getRotationMatrix2D(mid, angle, 1.0);
    warpAffine(gray, trans, handle, gray.size());
    equalizeHist(trans, trans);//直方图均衡化是通过拉伸像素强度分布范围来增强图像对比度的
一种方法
    resize(trans, trans, standardSize);
```

```
    trans.copyTo(eq);
    vect = eq.reshape(1, 1).t();//转置
    string name="";
    person_id = getIDfrompath(_imgpath);
}
void face::load_Eye(string path) {
    ifstream file(path.c_str());
    string s1, s2, s3, s4;
    string Read;
    if(!file){
        cout << "Can Load Eye file" << endl;
        return;
    }
    string line1;
    string line2;
    getline(file, line1);
    getline(file, line1);

    getline(file, line1);
    for (int i = 0; i < line1.size(); i++) {
        if (line1[i] >= '0' && line1[i] <= '9') {
            eyelx = eyelx * 10 + line1[i] - '0';
        }
    }
    getline(file, line2);
    for (int i = 0; i < line2.size(); i++) {
        if (line2[i] >= '0' && line2[i] <= '9') {
            eyely = eyely * 10 + line2[i] - '0';
        }
    }
    getline(file, line1);
    getline(file, line1);
    getline(file, line1);
    for (int i = 0; i < line1.size(); i++) {
        if (line1[i] >= '0' && line1[i] <= '9') {
            eyerx = eyerx * 10 + line1[i] - '0';
        }
    }
    getline(file, line2);
    for (int i = 0; i < line2.size(); i++) {
        if (line2[i] >= '0' && line2[i] <= '9') {
            eyery = eyery * 10 + line2[i] - '0';
        }
    }
    file.close();


}
void facelib::load(string pathi,string pathe) {
    vector<String> img1, img2, img3, img4, img5,img;
    vector<String> eye1, eye2, eye3, eye4, eye5,eye;
    glob(pathi + "/1.pgm", img1, true);
    glob(pathe + "/1.json", eye1, true);
    glob(pathi + "/2.pgm", img2, true);
    glob(pathe + "/2.json", eye2, true);
```

```
    glob(pathi + "/3.pgm", img3, true);
    glob(pathe + "/3.json", eye3, true);
    glob(pathi + "/4.pgm", img4, true);
    glob(pathe + "/4.json", eye4, true);
    glob(pathi + "/5.pgm", img5, true);
    glob(pathe + "/5.json", eye5, true);

    img.insert(img.end(), img1.begin(), img1.end());
    img.insert(img.end(), img2.begin(), img2.end());
    img.insert(img.end(), img3.begin(), img3.end());
    img.insert(img.end(), img4.begin(), img4.end());
    img.insert(img.end(), img5.begin(), img5.end());

    eye.insert(eye.end(), eye1.begin(), eye1.end());
    eye.insert(eye.end(), eye2.begin(), eye2.end());
    eye.insert(eye.end(), eye3.begin(), eye3.end());
    eye.insert(eye.end(), eye4.begin(), eye4.end());
    eye.insert(eye.end(), eye5.begin(), eye5.end());

    if (img.size() == 0 || eye.size() == 0) {
        cout << "no such files" << endl;
        return;
    }
    else if (img.size() != eye.size()) {
        cout << "imgs and eyes are not fit" << endl;
        return;
    }

    sort(img.begin(), img.end());
    sort(eye.begin(), eye.end());

    Size = img.size();
    for (int i = 0; i < img.size(); i++) {
        face *f = new face();
        f ->load_Img(img.at(i), eye.at(i));
        entrys.push_back(f);
        sam.push_back(f->trans);
        _samples.push_back(f->vect);//列向量
    }
    hconcat(_samples, samples);
    cout << "Size of facelib is " << Size << endl;
    Mat m;
    samples.convertTo(m, CV_8U);
    imwrite("face0_trans.png", entrys.at(0)->trans);
}
```

```
//mytest.cpp
#include"facelib.h"
extern facelib fl;


void mytest() {
    string model_name = "model";
    cout << "Read in Model..." << endl;
    FileStorage model(model_name, FileStorage::READ);
```

```
    Mat A;
    vector<Mat>allFaces;
    vector<Mat>src;
    vector<Mat>dst;
    int todet;
    int eigenCount;
    model["A"] >> A;
    model["allFaces"] >> allFaces;
    model ["eigenCount"] >> eigenCount;
    cout << "input the test object:" << endl;
    cin >> todet;
    for (int i = 6; i <= 10; i++) {
        string ima_path = "att-face/s" + to_string(todet) + "/" + to_string(i) +
".pgm";
        string eye_path = "att-eye-location/s" + to_string(todet) + "/" +
to_string(i) + ".json";
        face* f = new face();
        f->load_Img(ima_path, eye_path);
        Mat testImg = f->trans;
        src.push_back(testImg);
        Mat testDoubleMat;
        testImg.reshape(0, 1).convertTo(testDoubleMat, CV_64F);
        Mat testCoordinates = testDoubleMat * A;
        double minDistance = -1;
        face resultFace;
        int id = 0;

        for (auto iter = allFaces.begin(); iter != allFaces.end(); iter++, id++)
{
            double distance;
            face f_test;
            f_test.trans = *iter;
            Mat doubleMat;
            f_test.trans.reshape(0, 1).convertTo(doubleMat, CV_64F);
            f_test.eq = doubleMat * A;
            distance = 0;
            for (int i = 0; i < eigenCount; i++) {
                distance += pow(f_test.eq.at<double>(0, i) -
testCoordinates.at<double>(0, i), 2);
            }
            if (distance < minDistance || minDistance == -1) {
                minDistance = distance;
                resultFace = f_test;
            }
        }
        dst.push_back(resultFace.trans);
    }
    cout << "haha" << endl;
    Mat m1, m2, m;
    vector<Mat> fi;
    vconcat(src, m1);
    vconcat(dst, m2);
    cout << "haha" << endl;
    fi.push_back(m1);
    fi.push_back(m2);
```

```
    hconcat(fi, m);
    imwrite("test result.png", m);
    imshow("test", m);
    waitKey(0);
}
```

```
//myreconstruct.cpp
#include"facelib.h"
extern facelib fl;
void myreconstruct() {
    string model_name = "model";
    cout << "Read in Model..." << endl;
    FileStorage model(model_name, FileStorage::READ);
    Mat A;
    Mat eigenVectors = Mat();
    Mat mean;
    vector<Mat> allFaces;
    vector<Mat> tmp;
    Mat res;
    int eigenCount;
    model["A"] >> A;
    model["allFaces"] >> allFaces;
    model["eigenCount"] >> eigenCount;
    model["eigenVectors"] >> eigenVectors;
    model["mean"] >> mean;
    face* f = new face();
    int todet;
    int ph;
    cout << "input the person_id" << endl;
    cin >> todet;
    cout << "input the photo_id" << endl;
    cin >> ph;
    string ima_path = "att-face/s" + to_string(todet) + "/" + to_string(ph) +
".pgm";
    string eye_path = "att-eye-location/s" + to_string(todet) + "/" +
to_string(ph) + ".json";
    f->load_Img("att-face/s0/1.pgm", "att-eye-location/s1/1.json");
    Mat testDoubleMat;
    Mat testImg = f->trans;

    testImg.reshape(0, 1).convertTo(testDoubleMat, CV_64F);
    for (int i = 0; i < 5 ;i++) {
        int num_components;
        if (i == 0) num_components = 10;
        else if (i == 1) num_components = 25;
        else if (i == 2) num_components = 50;
        else if (i == 3) num_components = 75;
        else num_components = 585;
        Mat evs = Mat(A, Range::all(), Range(0, num_components));
        Mat testCoordinates = testDoubleMat * evs;
        Mat recst = testCoordinates * (evs.t());
        normalize(recst.reshape(1, f->trans.rows), recst, 255, 0, NORM_MINMAX);
        recst.convertTo(recst, CV_8UC1);
        tmp.push_back(recst);
    }
```

```
    hconcat(tmp, res);
    imwrite("reconstruct.png", res);
    imshow("res", res);
    waitKey(0);
}
```

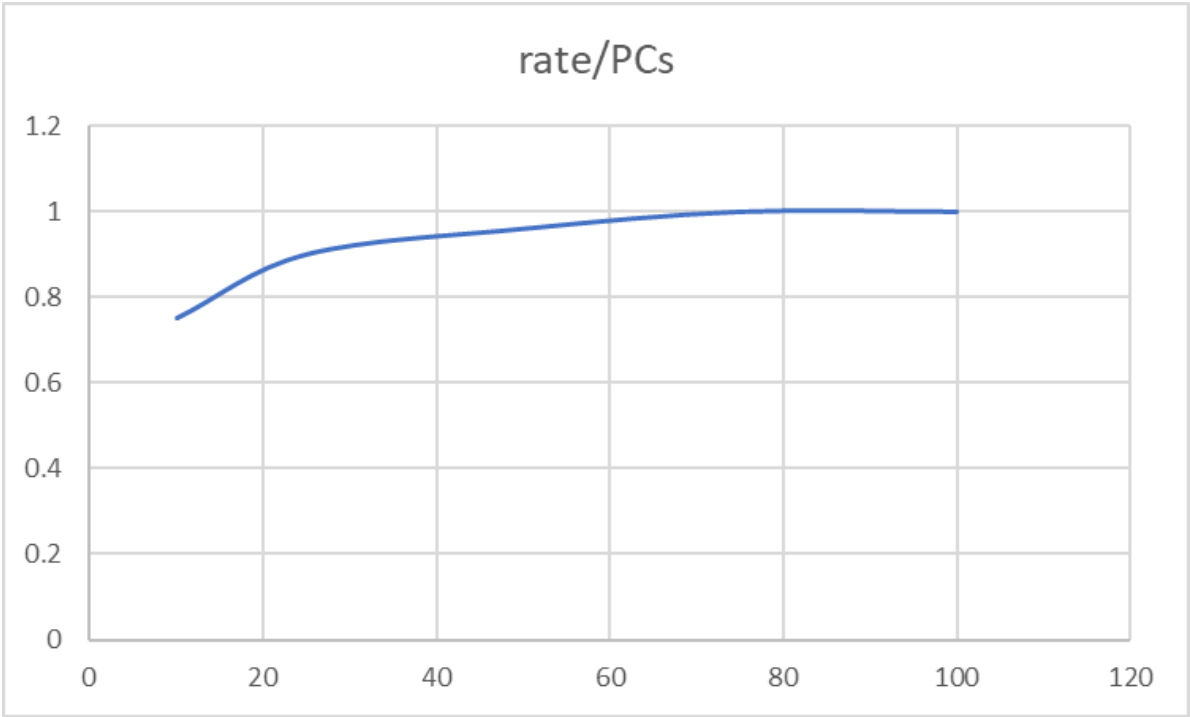
效果截图：

前十张特征脸：

将人脸数据库的后五张作为测试集，右侧是得出最相似的图片：

用10个PCs、25个PCs、50个PCs、75个PCs 以及100个PCs重构自己人脸：

Rank-1识别率的变化曲线



六、 结论与心得体会

本次实验实现了人脸检测功能，对于我来说，这个作业十分具有挑战性的，首先花了很大的精力理清了算法流程，在编程实现的时候，又被各种矩阵乘法错误困扰，以及什么时候需要变化矩阵形状，什么时候该使用转置矩阵。在利用了将近一周的时间做完本次作业后，我还是感觉非常有收获的。但是最终的绘图结果并没有非常的理想，可能是因为我在读取时进行了图片的压缩处理（为了减少计算溢出错误），我在后续的学习中也将对此进行优化。

七、 参考文献

博客-Eigenface(PCA)人脸识别

博客-EigenFace的原理、实现及性能评估